LAB 2

**Particle Swarm Optimization:**

Particle Swarm Optimization (PSO) is inspired by the social behavior of birds flocking or fish schooling. PSO is used to find optimal solutions by iteratively improving a candidate solution with regard to a given measure of quality.

CODE:

import numpy as np

# Define the Rastrigin function to optimize

def rastrigin(x):

A = 10

return A \* len(x) + sum((xi \*\* 2 - A \* np.cos(2 \* np.pi \* xi)) for xi in x)

# Particle class to represent each particle in the swarm

class Particle:

def \_\_init\_\_(self, dim):

self.position = np.random.rand(dim) \* 20 - 10 # Initialize position

self.velocity = np.random.rand(dim) \* 2 - 1 # Initialize velocity

self.best\_position = np.copy(self.position) # Personal best position

self.best\_value = rastrigin(self.position) # Personal best value

# Particle Swarm Optimization function

def pso(num\_particles, dimensions, num\_iterations, inertia\_weight, cognitive\_coef, social\_coef):

# Initialize particles

swarm = [Particle(dimensions) for \_ in range(num\_particles)]

# Initialize global best

global\_best\_position = swarm[0].best\_position

global\_best\_value = swarm[0].best\_value

for particle in swarm:

if particle.best\_value < global\_best\_value:

global\_best\_value = particle.best\_value

global\_best\_position = particle.best\_position

# Main PSO loop

for iteration in range(num\_iterations):

for particle in swarm:

# Update particle velocity

r1, r2 = np.random.rand(dimensions), np.random.rand(dimensions)

particle.velocity = (inertia\_weight \* particle.velocity +

cognitive\_coef \* r1 \* (particle.best\_position - particle.position) +

social\_coef \* r2 \* (global\_best\_position - particle.position))

# Update particle position

particle.position += particle.velocity

# Evaluate fitness

fitness\_value = rastrigin(particle.position)

# Update personal best

if fitness\_value < particle.best\_value:

particle.best\_value = fitness\_value

particle.best\_position = np.copy(particle.position)

# Update global best

if fitness\_value < global\_best\_value:

global\_best\_value = fitness\_value

global\_best\_position = np.copy(particle.best\_position)

# Optionally print progress

print(f"Iteration {iteration + 1}/{num\_iterations}, Best Value: {global\_best\_value}")

return global\_best\_position, global\_best\_value

# Parameters

num\_particles = 30

dimensions = 2 # Number of dimensions of the function

num\_iterations = 30

inertia\_weight = 0.5

cognitive\_coef = 1.5

social\_coef = 1.5

# Run PSO

best\_position, best\_value = pso(num\_particles, dimensions, num\_iterations, inertia\_weight, cognitive\_coef, social\_coef)

print(f"Best Position: {best\_position}")

print(f"Best Value: {best\_value}")

